从glibc源码看堆漏洞利用原理

付烨齐 2021年10月19日

PWN

- PWN的中文意义大概可以理解为二进制漏洞挖掘和利用，常见的二进制漏洞有缓冲区溢出（栈溢出，堆溢出，bss段溢出等），格式化字符串，条件竞争，整数溢出&截断，数组越界/Out Of Bound(oob)，空间复用，状态检查缺失（各类UAF都可以归为此类），内存未初始化，未检查函数返回值，库函数自身不严谨，程序自身设计不严密导致的逻辑漏洞等。

- PWN是CTF中一个比较困难的方向，前期的PWN以普通的Linux下C语言编写的glibc程序为主，后期深度和广度都会快速增长，广度可以扩展到windows/MacOS平台 && 其他编译型语言（C++/Go/Rust/wasm）下，深度可以扩展到windows/Linux/MacOS内核安全 && 虚拟化安全 && 浏览器安全领域等，也可以和逆向结合做一些其他好玩的事情。学好/精通二进制漏洞挖掘是一件需要时间和经验积累的事情。

堆利用

- 堆利用是学习glibc的一大难点，因为它考察的是程序漏洞和glibc堆管理代码（ptmalloc2）的配合，想要利用堆漏洞就必须熟悉glibc的堆的分配原理。

- 网上有关ptmalloc2的技术文章有很多，也有很多人画了各种流程图，但是这些总归是别人对源码的理解，难免有偏差和疏漏，而且有时候想要了解一些细节偏偏没人写过相关的分析文章。

- 所以今天的分享则是想以ptmalloc2相关的源代码作为切入点，带大家了解glibc堆管理器，以及一个经典堆attack的原理。

数据结构

- ptmalloc采用bin来对空闲的chunk进行管理：
- glibc2.23 fastbin ； unsorted bin；small bin；large bin；
- glibc 2.26后引入了 tcache

- 使用fastbinY数组来维护fast bin
- ptmalloc使用bins数组来维护small unsorted large bin：
- bins[1]为unsorted bin
- bins[2-63]为small bin
- bins[64-126]为large bin

数据结构

- malloc_chunk
- malloc_state

```
1110
1111  struct malloc_chunk {
1112
1113    INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
1114    INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */
1115
1116    struct malloc_chunk* fd;         /* double links -- used only if free. */
1117    struct malloc_chunk* bk;
1118
1119    /* Only used for large blocks: pointer to next larger size.  */
1120    struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
1121    struct malloc_chunk* bk_nextsize;
1122  };
```

```
1269  /* size field is or'ed with PREV_INUSE when previous adjacent chunk in use */
1270  #define PREV_INUSE 0x1
1271
1272  /* extract inuse bit of previous chunk */
1273  #define prev_inuse(p)       ((p)->size & PREV_INUSE)
1274
1275
1276  /* size field is or'ed with IS_MMAPPED if the chunk was obtained with mmap() */
1277  #define IS_MMAPPED 0x2
1278
1279  /* check for mmap()'ed chunk */
1280  #define chunk_is_mmapped(p) ((p)->size & IS_MMAPPED)
1281
1282
1283  /* size field is or'ed with NON_MAIN_ARENA if the chunk was obtained
1284     from a non-main arena.  This is only set immediately before handing
1285     the chunk to the user, if necessary.  */
1286  #define NON_MAIN_ARENA 0x4
1287
1288  /* check for chunk from non-main arena */
1289  #define chunk_non_main_arena(p) ((p)->size & NON_MAIN_ARENA)
```

```
1686  struct malloc_state
1687  {
1688    /* Serialize access.  */
1689    mutex_t mutex;
1690
1691    /* Flags (formerly in max_fast).  */
1692    int flags;
1693
1694    /* Fastbins */
1695    mfastbinptr fastbinsY[NFASTBINS];
1696
1697    /* Base of the topmost chunk -- not otherwise kept in a bin */
1698    mchunkptr top;
1699
1700    /* The remainder from the most recent split of a small request */
1701    mchunkptr last_remainder;
1702
1703    /* Normal bins packed as described above */
1704    mchunkptr bins[NBINS * 2 - 2];
1705
1706    /* Bitmap of bins */
1707    unsigned int binmap[BINMAPSIZE];
1708
1709    /* Linked list */
1710    struct malloc_state *next;
1711
1712    /* Linked list for free arenas.  Access to this field is serialized
1713       by free_list_lock in arena.c.  */
1714    struct malloc_state *next_free;
1715
1716    /* Number of threads attached to this arena.  0 if the arena is on
1717       the free list.  Access to this field is serialized by
1718       free_list_lock in arena.c.  */
1719    INTERNAL_SIZE_T attached_threads;
1720
1721    /* Memory allocated from the system in this arena.  */
1722    INTERNAL_SIZE_T system_mem;
1723    INTERNAL_SIZE_T max_system_mem;
1724  };
```

__libc_malloc

- malloc就是__libc_malloc
-
```
5302    strong_alias (__libc_malloc, __malloc) strong_alias (__libc_malloc, malloc)
```

- 申请时按照以下顺序查找空闲chunk：
- fast bin(LIFO,单向链表)
- small bin(FIFO,双向链表)
- unsorted bin(FIFO,双向链表)
- large bin(FIFO,双向链表)
- top chunk
- syscall

- 源码分析 glibc/malloc/malloc.c

unsorted bin attack

- 原理
- Unsorted Bin Attack 被利用的前提：
  - 控制 Unsorted Bin Chunk 的 bk 指针

- Unsorted Bin Attack 可以达到的效果：
  - 实现修改任意地址值为一个较大的数值

- demo

- 后续利用：
- 改global_max_fast
- house of orange

```
while ((victim = unsorted_chunks (av)->bk) != unsorted_chunks (av))
       {
         bck = victim->bk;
         ...
         }

         unsorted_chunks (av)->bk = bck;
         bck->fd = unsorted_chunks (av);
```

```
[ SOURCE (CODE) ]
In file: /pwn/unsortedbin_attack.c
   39          (void *)p[1]);
   40
   41  //------------------------------------
   42
   43  malloc(0x190);
 ► 44  fprintf(stderr, "Let's malloc again to get the chunk we just free. During "
   45                  "this time, target should has already been "
   46                  "rewrite:\n");
   47  fprintf(stderr, "%p: %p\n", &target_var, (void *)target_var);
   48 }
   49
[ STACK ]
00:0000| rsp  0x7ffd8c976a00 —▸ 0x400890 (__libc_csu_init) ◂— push   r15
01:0008|      0x7ffd8c976a08 —▸ 0x7fbe17e7ab78 (main_arena+88) —▸ 0x193e3b0 ◂— 0x0
02:0010|      0x7ffd8c976a10 —▸ 0x193e010 —▸ 0x7fbe17e7ab78 (main_arena+88) —▸ 0x193e3b0 ◂— 0x0
03:0018|      0x7ffd8c976a18 ◂— 0x689a94cc2ff2600
04:0020| rbp  0x7ffd8c976a20 —▸ 0x400890 (__libc_csu_init) ◂— push   r15
05:0028|      0x7ffd8c976a28 —▸ 0x7fbe17ad6840 (__libc_start_main+240) ◂— mov    edi, eax
06:0030|      0x7ffd8c976a30 ◂— 0x0
07:0038|      0x7ffd8c976a38 —▸ 0x7ffd8c976b08 —▸ 0x7ffd8c978966 ◂— '/pwn/a.out'
[ BACKTRACE ]
 ► f 0           400828 main+386
   f 1     7fbe17ad6840 __libc_start_main+240

pwndbg> p &target_var
$1 = (unsigned long *) 0x7ffd8c976a08
pwndbg> p target_var
$2 = 140454421572472
pwndbg>
$3 = 140454421572472
pwndbg> p/x target_var
$4 = 0x7fbe17e7ab78
pwndbg>
```

glibc2.29

- 增加了检测逻辑

```
while ((victim = unsorted_chunks (av)->bk) != unsorted_chunks (av))
       {
         bck = victim->bk;
         if (__glibc_unlikely (bck->fd != victim)|| __glibc_unlikely (victim->fd != unsorted_chunks (av)))
         malloc_printerr ("malloc(): unsorted double linked list corrupted");
```

- unsorted bin attack失效
- 仍有其他方法实现向指定地址写一个大数的效果。
- 例如 largebin attack

glibc2.xx

- glibc2.31 ubuntu20.04 发行版使用

- 2022年 ubuntu22.04发行 glibc2.35上的堆利用 => 新保护机制，新的绕过手段。